Lattice-gas hydrodynamics on the IBM 3090 Vector Facility

by S. Succi
D. d'Humières
F. Szelényi

After a brief review of the means for characterizing lattice gases using cellular automata rules, we discuss the implementation of the rules for simulating hydrodynamic phenomena which can be described by the Navier–Stokes equations. Special emphasis is placed on data-mapping strategies and implementation through the use of the high speed and large memory resources offered by vector multiprocessors such as the IBM 3090 Vector Facility. We present performance data which pertain to square and hexagonal lattice gases, and discuss the limits of the approach used and its potential extendability to other areas.

Introduction

Lattice-gas models which obey cellular automata (CA) rules are of increasing interest in connection with the simulation of complex hydrodynamic phenomena [1]. By using appropriate restrictions on the crystallographic symmetries of the lattice, and by taking appropriate limits, various fluid-dynamics equations are obtained which describe the macroscopic behavior of the lattice gas as a continuum system despite its underlying discrete nature. Since the cellular automata rules are intrinsically discrete and involve only logical operations, a new simulation strategy is offered which is qualitatively different from the "conventional" approximation procedures usually adopted for the solution of partial differential equations. From the computational point of view, the key aspect of CA rules is the use of quantized variables, i.e., variables which take values only in a small set of integers, typically just zero or one. This offers the possibility of organizing the data structure in such a way as to code only one bit per dynamic variable, as opposed to "conventional" simulations based on real (floating-point) variables requiring an entire computer word (32, 64 bits). Clearly, one cannot expect a single CA variable to capture the same amount of information as a 32- or 64-bit floating-point variable. However, one should bear in mind that, in contrast to the floating-point representation in which the least significant bits carry progressively less information, in a CA simulation all bits have the same importance. Moreover, since integers enjoy an exact representation in the computer, CA simulations are by definition free of the approximation (truncations, round-off) errors which affect simulations based on real variables. Therefore, it is reasonable to expect that the CA approach should permit the processing of the same amount of useful information, and retain it in the course of the calculation at a lower cost in CPU time and storage than a conventional simulation. In this paper, we present and discuss the criteria which govern the efficient implementation of CA rules in a vector multiprocessor such as the IBM 3090 Vector Facility. Performance data are presented, and some qualitative elements of comparisons with other techniques commonly adopted for the solution of

the Navier–Stokes equations are discussed. The programs developed thus far are concerned only with lattice-gas models simulating the Navier–Stokes equations in two dimensions. Extensions to three dimensions are discussed briefly. Applicability to other areas, such as geophysics, magnetohydrodynamics, etc. is not covered in this paper. The reader interested in such applications is referred to the reference list given in [1] and the papers included in [2].

Basic principles of lattice-gas automata

In this section we briefly review the basic elements of the theory of discrete lattice gases; since a complete discussion of the theory is beyond the scope of this paper, the reader interested in an exhaustive treatment of this subject is referred to [1, 3, 4].

We begin by considering a two-dimensional Bravais lattice Λ consisting of L^2 spatial sites. It is assumed that each of the sites can hold up to S particles, endowed with unit mass and unit speed along the directions defined by the links connecting each site to its S neighbors (S is the coordination number of the lattice). The microscopic configuration of the system is thus defined by a set of positions $\vec{r}_i \in \Lambda$, a set of unit speeds $\vec{c}_i^s \equiv \vec{c}_s$, and corresponding occupation numbers $N_i^s \in Z^{2S} \equiv (0, 1)^S$, $i = 1, L^2$, $s = 1, S$, which assume the values of 0 or 1 depending on whether or not the ith site is occupied by a particle pointing in the direction s. For example, in the four-link lattice gas introduced by Hardy, Pomeau, and De Pazzis [3], henceforth designated as HPP,

$$\vec{c}_1 = (1, 0), \quad \vec{c}_2 = (0, 1),$$
$$\vec{c}_3 = (-1, 0), \quad \vec{c}_4 = (0, -1). \tag{1}$$

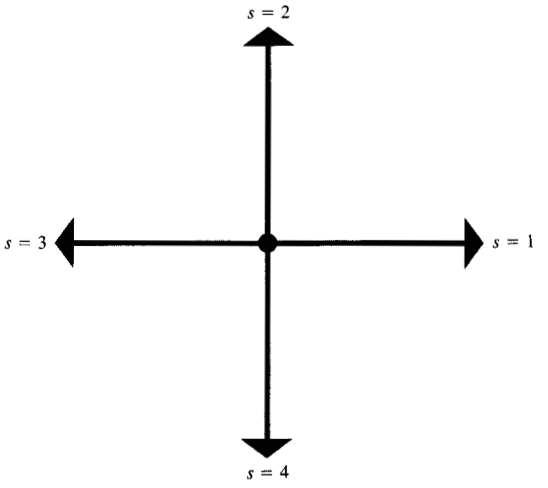

Figure 1

Four links emerging from a gridpoint of an HPP lattice gas.

Illustrated in **Figure 1** is a gridpoint of an HPP lattice gas and four associated emerging links. In **Figure 2**, the row vector $|0000\rangle$ indicates the total absence of particles (hole); the row vector $|0101\rangle$ indicates a half-occupied site, and the row vector $|1111\rangle$ indicates a fully occupied site.

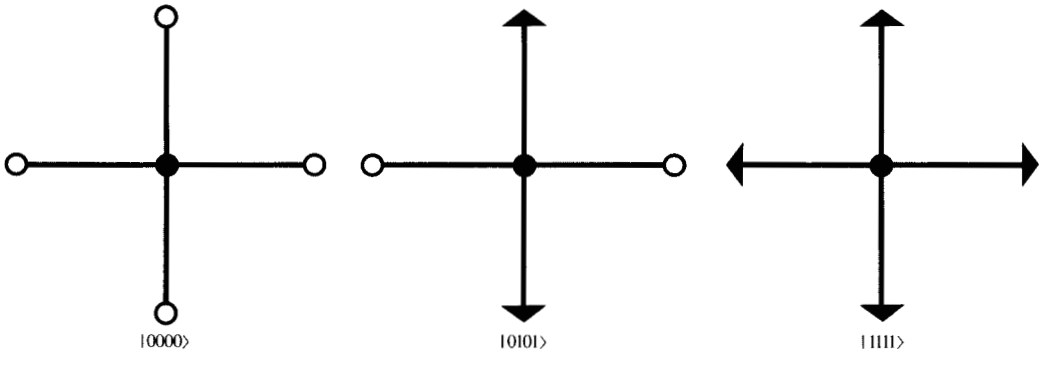

Figure 2

Row vectors, indicating (from left to right) a hole, a half-occupied site, and a fully occupied site in an HPP lattice gas.

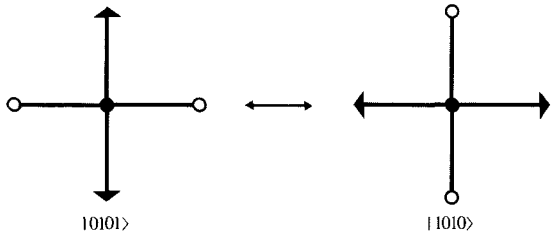

Figure 3

Collision configurations of the HPP lattice gas. Zeros indicate the absence of a particle and ones indicate its presence.

Since the bits are indistinguishable with regard to their importance, the corresponding particles obey an exclusion principle which forbids the simultaneous presence of two or more of them at a given site if their velocities are in the same direction. Thus, in a pictorial sense, we can view the automaton as a Fermionic gas of bits in mutual interaction.

The evolution of the automaton consists of two phases: a free-streaming phase, in which all the particles propagate synchronously one step ahead along the link associated with their speeds, and a collision phase, in which at each node the particle speeds are redistributed according to simple rules whose details depend on the type of automaton under consideration, although they are in any case subject to constraints of conservation (typically, particle number and linear momentum).

Collisions are assumed to be instantaneous, and the free-streaming of the particles is assumed to take place between two distinct instants t and $t + 1$ (time is also a discrete variable). Consequently, the evolution of the Boolean field N_i^s is governed by a first-order difference equation (Liouville equation) of the form

$$\Delta_t N_i^s = \mathcal{L} N_i^s, \tag{2}$$

where Δ_t is the discrete forward-time-differencing operator and $\mathcal{L}$ the Liouville evolution operator.

The Liouville operator is given by the direct product of a free-streaming operator $\mathcal{I}$ and a collision operator $\mathcal{C}$, defined as follows:

$$\mathcal{C}: N_i^s(t) \rightarrow N_i^s(t) + \delta^s(N_i^1, \cdots, N_i^S),$$

$$\mathcal{I}: N_i^s(t) \rightarrow N_{i+\sigma(s)}^s(t + 1), \tag{3}$$

where σ is the increment of the index i in the direction of the vector $\vec{c}_s$ and δ is a function that indicates the change of the particle population at a given node due to the collision interaction.

At each site there are 2^S possible configurations, and the collision operator is a mapping from Z^{2S} to Z^{2S}, which reduces to the identity for all those configurations which cannot be altered without violating the aforementioned conservation laws and exclusion principle. In the remaining cases, the collision operator decreases/increases (annihilation/generation) the occupation number of the site by one unit. For example, in the case of the four-link HPP automaton, it is easy to verify that only two out of the sixteen (2^4) possible configurations can give rise to a legal collision; these are the doublets $|0101\rangle$ and $|1010\rangle$, which transform into one another by a rigid rotation of $\pi/2$ (see **Figure 3**).

It is easy to verify that in the case of an HPP lattice gas, the collision function δ^s takes the following simple form (the spatial index i is omitted for the sake of simplicity, and $N_s \equiv N_i^s$):

$$\delta^s = N_{s+1}N_{s+3} \times (1 - N_s)(1 - N_{s+2}) - N_sN_{s+2} \times (1 - N_{s+1})(1 - N_{s+3}), \tag{4}$$

where the subscripts are to be taken modulo 4. The HPP automaton is deterministic in that the correspondence between the precollision and postcollision configurations is one-to-one. When the coordination number is increased, this is no longer true; consequently, the collision operator becomes stochastic, in the sense that the final state must be chosen randomly from a set of permitted final configurations.

The microscopic behavior of the lattice gas can be analyzed using the standard techniques of statistical mechanics [5]; in particular, analysis of the equilibrium states shows that the Liouville distribution function is factorized in terms of single-node distribution functions. As a result, the mean occupation numbers

$$n_s = \sum_{i=1}^{L^2} \frac{N_i^s}{L^2} \tag{5}$$

are distributed according to the Fermi–Dirac law

$$n_s = \frac{1}{1 + \exp(h + \vec{q} \cdot \vec{c}_s)}, \tag{6}$$

where the scalar h and the vector $\vec{q}$ are functions of the mean density ρ and macroscopic velocity $\vec{u}$, defined as

$$\rho = \sum_{s=1}^{S} n_s, \qquad \vec{u} = \sum_{s=1}^{S} n_s\vec{c}_s. \tag{7}$$

At thermodynamic equilibrium, the macroscopic density ρ and speed $\vec{u}$ become constant in space. To understand how this equilibrium is attained, one must examine the transport processes which take place in the system. The macroscopic equations which govern these processes are obtained by postulating a local equilibrium distribution of the same form

as the one in Equation (6), where ρ and $\vec{u}$ are assumed to exhibit small-amplitude long-wavelength departures from their equilibrium values. One then expands Equation (6) around their equilibrium solution and subsequently adopts a Chapman–Enskong expansion [6] of the macroscopic speed. After retention of terms up to the second order, this leads to fluid-like equations which in two dimensions take the following form:

$$\partial_t \rho + \sum_{l=1}^{2} \partial_l \rho u_l = 0, \tag{8}$$

$$\partial_t \rho u_l + \sum_{m=1}^{2} \partial_m (P_{lm} - \nu Q_{lm}) = 0, \tag{9}$$

where

$$P_{lm} = \rho \left[G(\rho) \sum_{n,o=1}^{2} T_{lmno} u_n u_o + \frac{c^2}{2} \delta_{lm} \right],$$

$$Q_{lm} = \rho \sum_{n,o=1}^{2} T_{lmno} \partial_n u_o, \tag{10}$$

and

$$G(\rho) = \frac{2}{Sc^4} \frac{S - 2\rho}{S - \rho}, \tag{11}$$

$$T_{lmno} = \sum_{s=1}^{S} c_{sl} c_{sm} U_{sno}, \tag{12}$$

$$U_{sno} = c_{sn} c_{so} - \frac{c^2}{2} \delta_{no}. \tag{13}$$

The coefficient ν in Equation (9) is related to the kinematic viscosity of the fluid, which is responsible for the dissipative effects taking place in the lattice gas and is crucially related to the details of the collision operator.

The above expressions are closely related to the Navier–Stokes equations except for the factor $G(\rho)$ and the tensor T_{lmno}, which stem from the intrinsic discreteness of the underlying lattice. In particular, depending on the geometry of the lattice, these factors cause breaking of the translational (Galilean) and rotational invariance (isotropy) at a macroscopic level. Although Galilean invariance is recovered in the continuum limit, this is not generally the case for rotational invariance, since the group of rotations (approximated by Z^S) is compact. In fact, following the arguments presented in [1], it is possible to show that in order for the momentum flux tensor P_{lm} and the viscous tensor Q_{lm} to be isotropic, the underlying lattice must exhibit at least a sixfold symmetry. It is this very observation, with the consequent introduction of the Frisch, Hasslacher, and Pomeau automaton [4], henceforth designated as FHP, which has stimulated interest in using lattice gases to investigate hydrodynamic problems.

As usual, in order for the fluid equations to be a self-contained set, a closure condition (state equation) is required to fix a further relation among the thermodynamic variables ρ, $\vec{u}$, and p (scalar pressure). This third relation is indeed hidden in Equations (10), which implicitly embody the state equation

$$p = constant \times \rho. \tag{14}$$

This equation highlights the fact that, owing to the dispersionless nature of the velocity distribution function, the thermodynamic notion of temperature is missing in the lattice gas. Thermal effects can be introduced by allowing particles to move to the nearest neighbor (speed 1) and to the next-nearest neighbor (speed $\sqrt{3}$) in the lattice [7].

In summary, this brief survey of lattice gases indicates that they exhibit macroscopic properties which are similar in many respects to those of a real fluid, except for the density-dependent factor $G(\rho)$ in the nonlinear advective term and tensorial relationships. In spite of these undeniable physical drawbacks, lattice gases are of considerable interest because of their conceptual simplicity and the resulting potential advantages they offer from a computational point of view, especially in conjunction with a vector and parallel processing environment.

Implementation on a vector multiprocessor

Having reviewed the basic concepts underlying the theory of lattice-gas automata, we now discuss how the theory can be implemented and transformed into an efficient computational tool.

To implement CA rules efficiently on a vector multiprocessor, one must devise data-mapping strategies allowing it to take full advantage of the essential characteristics of the updating rule which governs the evolution of the lattice-gas automaton; these characteristics are *locality*, *uniformity*, and *speed quantization*. As usual, locality is regarded as implying that only a few neighbors interact, while uniformity is regarded as implying that the updating rule is the same for each dynamic variable regardless of its spatial and temporal location. As in any other computational context, these two properties ideally match the concepts of parallelism and vectorization, respectively. Because speed quantization is peculiar to CA simulations, it offers a third computational opportunity which has no counterpart in "conventional" fluid dynamics, although it definitely has one in statistical mechanics [8] (spin lattices). In fact, starting from the fact that speeds assume only the values of 0 or 1, one realizes that it must be possible to pack the data in such a way as to spend just one bit per dynamical variable (it is worth stressing that the necessity of packing data is a direct consequence of the fact that we need to work in a word-oriented environment). This is the philosophical essence of the method which, rather than being based on real (floating-point) variables, each requiring a whole computer word (say W bits, with $W = 32, 64$) of storage, is conceived to treat single bits as working units.

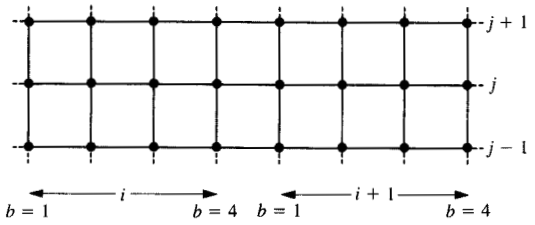

Figure 4

Data structure for the HPP lattice gas of Figure 1.

Returning to our problem, let us see how these ideas can be made practical.

As a first step we must distinguish between the two basic phases of the evolution, namely free propagation and collision. For the sake of simplicity let us refer to the simple case of the four-link lattice sketched in Figure 1. In this lattice there are only two independent displacement directions: horizontal (left/right) and vertical (up/down). If we decide to assign each of these links to a single bit of an integer variable, it follows that the Boolean field N_i^s can be represented in computer storage as a set of $S = 4$ two-dimensional arrays $I_s(i, j)$. If we further pack bits along the horizontal direction (x) (see **Figure 4**, for which $W = 4$), then the entire structure of the lattice is recovered by letting the index i vary between 1 and L/W and the index j between 1 and L.

Once this data structure has been specified, the Move operator can be formalized as follows:

$$I_s(i', j', b', t + 1) = I_s(i, j, b, t) \qquad s = 1, 4, \qquad (15)$$

where the triplet (i, j, b) indicates the bth bit in the subscript $I_s(i, j)$. If we choose I_1 to represent the bits which must move "right" ($s = 1$), the specific form of the Move rule is then

$$i' = i, \qquad j' = j, \quad b' = b + 1 \qquad (b = 1, W - 1) \qquad (16)$$

$$i' = i + 1, \quad j' = j, \quad b' = 1 \qquad (b = W), \qquad (17)$$

where bits are supposed to increase from left to right (in actual storage the numbering is the other way around). Displacements along the vertical direction are even simpler:

$$i' = i, j' = j + 1 \qquad (b = 1, 2, \cdots, W). \qquad (18)$$

Both of these mappings can easily be implemented with appropriate FORTRAN instructions. For vertical displacements, one simply needs to copy one vector onto another:

```
      DO 2 I=1,L/W
        DO 1 J=2,L
1         INEW(J,I) = IOLD(J-1,I)
2     INEW(1,I) = IOLD(L,I)
```

where OLD and NEW denote the instants t and $t + 1$, respectively.

This construct is optimal in the sense that it at best exploits the data packing (each copy operation simultaneously updates W dynamic variables), and vectorizes over contiguous memory locations (stride 1) along the maximal vector length L. Note that in order to achieve this latter feature, the unpacked index J has been designed to be the internal one.

When bits have been packed along the horizontal direction, the internal boundaries set up by the finiteness of the word length (b = W) are completely "transparent" to the vertical displacements; this results from the displacements being exactly orthogonal to the packing direction. Unfortunately, this is not the case for the horizontal displacements, which must necessarily run through the word boundaries. In fact, the mapping given by Equations (16) and (17) can be implemented via the FORTRAN routine ISHFT [9], which shifts the content of a single register by a prescribed number of bits according to the syntax

```
IS = ISHFT(I,m).
```

This means that the bits in I are shifted m positions ($-31 \le m \le 31$ with 32-bit-long words) right ($m < 0$) or left ($m > 0$) and placed into IS; bits shifted out of the vector register are lost and those shifted in are padded with zeros.

This is perfectly appropriate for the internal bits ($1 \le b < W$), but does not work for the boundary bits, which, instead of flowing in the leftmost positions of the next subscript, are simply lost from the register. This implies that the boundary bits must be saved prior to the shift operation and must subsequently be placed in the correct positions.

To date, the procedure we have adopted consists of three steps:

1. M = ISHFT(IOLD(I,J−1),−31)	Pick up the rightmost bit of IOLD(J,I−1) and place it into the leftmost bit of the mask M.
2. INEW(J,I) = ISHFT(IOLD(J,I),−1)	Move the subscript IOLD(J,I).
3. INEW(J,I) = IOR(INEW(J,I),M)	Force the leftmost bit of M into the leftmost bit of INEW.

The routine IOR is the FORTRAN routine used for performing the connective OR operation. Note that the last operation guarantees that the rightmost bit of M will be placed into the leftmost bit of INEW(J,I), since after the shift

this bit is padded with a zero. The above procedure involves three logical operations, two of which (the first and the last) are required only because of the boundary bit! This drawback is inherently related to word finiteness, i.e., to the fact that use is being made of a general-purpose computer. In this sense, the optimal situation would be to have longer computer words in order to minimize the L^2W^{-1} operations needed to sweep around the computational domain. The other extreme would be to resort to a set of L^2 one-bit independent processors, which characterizes special-purpose hardware [10]. Fortunately, as we see in the next section, the CPU cost due to the boundary-bit problem is not proportional to the number of extra operations, because the CPU time required by data transfer (fetch/store operands from/to storage) can be of the same order of magnitude as the time spent by the operands in the logical/arithmetical processing unit.

The discussion thus far has referred to the simplest case of a square-lattice automaton. Obviously, when the coordination number of the lattice is raised from 4 to 6, matters become considerably more involved. However, the basic principles remain essentially the same.

In the six-link lattice sketched in **Figure 5**, there are three independent directions of propagation, which we conveniently label as "right/left" ($s = 1, 4$); "up-right/down-left" ($s = 2, 5$) and "up-left/down-right" ($s = 3, 6$). Since there are now three propagation directions but only two macroscopic dimensions, it is clear that, no matter how the gridlines are numbered, we must account for diagonal displacements which directly couple the triplet (i, j, b) to the triplet $i, j \pm 1, b \pm 1$). For example, if we want to maintain the same data structure introduced for the square lattice, the Move "up-right" reads as follows:

$$j' = (j + 1)_L, \qquad i' = i, \qquad b' = b + 1 \qquad (b = 1, W - 1), \qquad (19)$$

$$j' = (j + 1)_L, \qquad i' = (i + 1)_{L/W}, \qquad b' = 1 \qquad (b = W), \qquad (20)$$

where the subscript designates "modulo." This is illustrated in **Figure 6**.

These transitions can be implemented by acting simultaneously upon the corresponding arrays with Copy and Shift operations, viz.,

```
INEW(J,I) = ISHFT(IOLD(J−1,I),1).
```

Of course, the boundary-bit problem also becomes a little more vexing, since these cross-diagonal hopping terms require the three-step procedure outlined above. Also, in general one must be careful in handling the boundary conditions, since there are bits which belong to all three types of boundaries, along X, Y, and b. However, it is worth pointing out that the conceptual scheme does not change

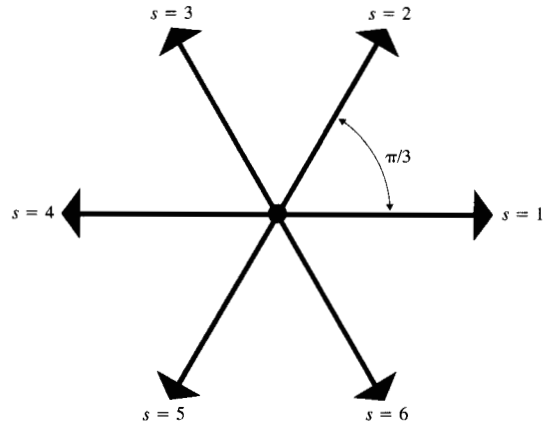

Figure 5

The six links emerging from a gridpoint in an FHP lattice gas.

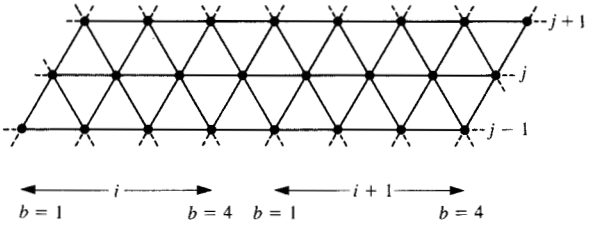

Figure 6

Data structure for the FHP lattice gas.

qualitatively, and no additional software is required to handle higher coordination numbers.

Having discussed the implementation of the free-streaming operator, we can now turn our attention to the collision phase. Assuming the same data structure as for the free-streaming phase, the collision step can be formalized as follows:

$$I_s(i, j, t) = \mathcal{B}_s[I_1(i, j, t) \cdots I_S(i, j, t)], \qquad (21)$$

where $\mathcal{B}_s$ is a suitable set of Boolean operators. Note that

since the collisions are ultralocal in space and time, the indices i, j, t on the left-hand and right-hand sides of the above equation are the same.

In practice, the Boolean operators $\mathcal{B}_s$ can be constructed either as a combination of elementary Boolean functions (OR, XOR, AND) or alternatively as a lookup table mapping the bits of the precollision state into those of the postcollision one. This latter solution causes a deterioration (by roughly a factor of S/W) of the performance of the free-streaming phase. In fact, since the S links emanating from a site must be coded over contiguous bits of the same word, each word contains only W/S bits which propagate along the same direction. As a result, only W/S bits can be moved simultaneously with a single operation. On the other hand, resorting to a lookup table becomes mandatory in applications such as those involving mixtures [12], in which the number of bits per site is so high that a closed Boolean expression of the collision operator cannot be found. The same consideration holds for hydrodynamics in three dimensions.

However, as long as one is interested in two-dimensional hydrodynamic applications, there is no question that the data structure introduced previously, namely S independent two-dimensional "layers" of bits, is to be preferred.

Once this organization has been stipulated, the collision phase can be implemented by means of the following single DO-loop (IEOR is the FORTRAN routine performing the exclusive-OR):

```
DO 1 I=1,L/W
 DO 1 J=1,L

     MASKC1 = BOOLE1(I1(J,I), · · · I6(J,I))
     MASKC2 = BOOLE2(I1(J,I), · · · I6(J,I))
     . . . . .
     MASKC6 = BOOLE6(I1(J,I), · · · I6(J,I))
     I1(J,I) = IEOR(MASKC1,I1(J,I))
     I2(J,I) = IEOR(MASKC2,I1(J,I))
     . . . . .
1    I6(J,I) = IEOR(MASKC6,I6(J,I))
```

where BOOLE1–BOOLE6 represent a set of Boolean relations constructed in such a way that the bits of the collision masks MASKC1–MASKC6 are ON or OFF depending on whether or not the corresponding site holds a collision configuration.

Again, we are dealing with an ideal construct which has the same advantageous properties of vectorizability and packing efficiency already discussed for the vertical displacements. Moreover, this construct is independent of the coordination number of the lattice, which affects only the degree of complexity of the Boolean expression needed to construct the collision masks. The coordination number has crucial consequences regarding the properties of the lattice gas, and obviously also an important impact on the resulting computational performance; however, it remains true that the above implementation structure is independent of these problems.

Before closing this section, we consider a few aspects of the correspondence of *uniformity* to *vectorization*, *speed quantization* to *data packing*, and *locality* to *parallelization*. We have already seen that the first two can be optimally achieved by means of appropriately chosen DO-loops. With respect to the parallelization, we first note that the lattice Λ is by construction expressible as the direct sum of S decoupled sublattices Λ_s, each corresponding to a different direction of propagation. If we imagine partitioning each of these sublattices into R subregions, we can write

$$\Lambda = \bigcup_{s=1}^{S} \Lambda_S = \bigcup_{s=1}^{S} \bigcup_{r=1}^{R} \Lambda_{sr} . \tag{22}$$

This expression indicates that two levels of parallelism are in principle available. The degrees of parallelism associated with these two levels, say P_1 and P_2, have upper bounds S and R, respectively. Note that by taking the limit $P_1 \rightarrow S$ and $P_2 \rightarrow R \rightarrow L^2$, the "special hardware" view of SL^2 locally connected one-bit processors is recovered.

For the Move step, to implement the second level of parallelism one must account for the data transfer between the different processors imposed by the existence of internal boundaries between the various sublattices. For instance, if we partitioned the computational domain into NP horizontal slices, the conceptual procedure to move bits "up" would read as follows (the DO-loop refers to the Pth processor):

```
      DO 1 I=1,L/W
1       INEW(1,I,P) = IOLD(L/NP,I,PM1)

      DO 2 I=1,L/W
        DO 2 J=2,L/NP
2         INEW(J,I,P) = IOLD(J−1,I,P)
```

where the index PM1 = P − 1 for P = 1, 2, · · · , NP − 1 and PM1 = 1 for P = NP.

Unlike the free-streaming phase, the collision step does not lend itself to the first level of parallelization because the "internal" speed states are by definition all coupled. However, the second type of parallelization is trivial because, since the collisions are ultralocal, each subdomain is completely decoupled from the others, and consequently no information needs to be passed through the boundaries. As a result, the parallelization procedure is reduced to a trivial segmentation of the corresponding DO-loop [13].

Performance data

The basic principles illustrated in the previous section have been implemented in a series of computer programs running on the IBM 3090 Vector Facility. In particular, three lattice-gas-automata codes have been developed thus far: HPP, and FHP Models 1 and 2, which respectively involve 4, 6, and 7 variables per site. In the last case, the seventh bit is used to

Table 1 CPU ms required to run one step of the HPP, FHP1, and FHP2 lattice-gas automata.

	Scalar mode	*Vector mode*
HPP	66	36
FHP1	120	69
FHP2	233	138

Table 2 Partition of the CPU ms/step in vector mode.

Type	*MOVE*	*MOVET*	*COLL*
HPP	7.1 + 4.0 + 7.1 + 4.0	22.2	14
FHP1	6.1 + 5.5 + 5.5 + 6.1 + 5.0 + 5.0	33.2	36
FHP2	6.1 + 5.5 + 5.5 + 6.1 + 5.0 + 5.0	33.2	105

Table 3 Partition of CPU ms/step in scalar mode.

Type	*MOVE*	*MOVET*	*COLL*
HPP	12 + 8.1 + 12 + 8.1	40.2	26
FHP1	10.2 + 8.3 + 8.3 + 10.2 + 8.3 + 8.3	53.6	66
FHP2	10.2 + 8.3 + 8.3 + 10.2 + 8.3 + 8.3	53.6	179

Table 4 CPU ms/step required by the Collision, Move, and Advance steps of an HPP lattice gas. The data are from [14].

	Scalar mode	*Vector mode*
COLLIDE	25	14
MOVE	46	23
ADVANCE	21	09

introduce "rest" particles which participate only in the collision phase without moving. It can be shown [1] that the inclusion of these particles significantly affects the transport coefficients of the fluid, and in particular causes a significant reduction of kinematic viscosity, which leads to an increase in the highest Reynolds number achievable in the simulation.

All of the data presented refer to a single processor and a 1024×1024 grid (the CPU time to perform one time-step scales linearly with the total number of points in the lattice).

A typical set of CPU times required to run one step of the three different automata is shown in **Table 1**. Equivalently, one second of CPU time is required for about 30 sweeps over a 1024×1024 HPP lattice and about 7 sweeps over a 1024×1024 FHP2 lattice, referred to as 30 and 7 megasites per second, respectively.

In each case, use of the vector mode results in an increase by about a factor of 2 over the scalar mode, at virtually no cost in terms of programming effort. Table 1 also shows the increasing cost of CPU time as the complexity of the automaton is raised. Comparing the HPP and FHP1 cases, this is due to both the Move and Collision steps. Comparing the FHP1 and FHP2 cases, the increased cost is caused entirely by the collisions (rest particles do not move!), which require more Boolean algebra.

This is more evident in **Table 2**, which contains a more detailed partition of the CPU ms/step among the various stages of the evolution, as obtained by running the programs in the vector mode. The numbers appearing under the *MOVE* column refer to the different directions of propagation, numbered counterclockwise, as in Figures 1 and 5.

The entries in the *MOVET* (total move time) column of Table 2 proceed almost linearly with coordination number, which is 4 for the HPP lattice and 6 for the FHP1 and FHP2 lattices. The numbers in the *COLL* (collision) column reflect the increasing number of logic operations required to construct the respective operators, viz., 9, 35, and 125 in the three cases. Note, however, that CPU ms/step scales less than linearly with the Boolean operations because of the cost of data transfer from memory to registers and vice versa.

The nonuniform partition in the *MOVE* column is a signature of the bit-boundary problem; in fact, with data packed horizontally, Move Right and Move Left are the most expensive steps. The same applies to the scalar mode, as indicated in **Table 3**.

We have already mentioned the importance of accounting for the cost of data transfer. An example is presented in **Table 4**, which cites the CPU ms per step required by a preliminary version of the HPP lattice-gas code in which the advancement of the time variable was performed in a separate routine (Advance) whose job was simply to fetch the OLD arrays and copy them onto NEW.

From the data of Table 4, it is evident that the cost of loading and storing is approximately the same as that required by the logical elaboration. Obviously, as the complexity of the collision operator is raised, this problem becomes less significant.

Actually, the time variable can be advanced without introducing two separate sets of arrays for the times t and $t + 1$. However, this requires some effort to ensure that only those variables which are no longer needed for the Move step undergo the time advancement; differently phrased, time recurrences must be avoided. Clearly, this can be achieved by scanning the DO-loops upstream.

Again, extra care is necessary regarding the boundaries, which must be treated with the aid of temporary arrays holding the content of the boundary bits, in accordance with the procedure described in [14]. This approach is very effective, because it gains all of the CPU time required by the Advance step; above all, it yields a saving of a factor of 2 in required computer storage. The data reported in Tables 1–3

Table 5 Elapsed time (ms/step) required by the Move and Collision steps in vector mode as a function of the number of processors used.

P	*MOVE*	*COLL*	*SPEEDUP*
1	23.6	99.0	1
2	12.5	50.0	1.96
3	9.2	34.5	2.80
4	9.1	25.7	3.52
5	8.2	22.1	4.05
6	4.7	19.5	5.06

refer to a version of the programs in which this rule was used, although some further means related to the optimal usage of vector registers (such as resorting to temporary scalars holding the contents of subscripts) were not used. Use of the latter was found to yield a gain in CPU time by about a factor of 1.3. Further gains can be contemplated: For example, the word-boundary problem might be alleviated by coding adjacent sites as homologous bits of contiguous words instead of contiguous bits of the same word. Consider coding sites 1 to L/W in the leftmost bit (the Wth) of the elements $IR(1), \cdots, IR(L/W)$, sites $L/W + 1$ to $2L/W$ in the $(W - 1)$th bit and so on up to the rightmost bit ($b = 1$). In this case, if IR represents the bits which must move "right," the propagation is achieved by simply copying the element $IR(I)$ into $IR(I + 1)$. The internal boundaries imposed by the finiteness of the word length would occur at each L/W site instead of at each W site as in the present implementation. This is convenient whenever $L > W^2$; i.e., $L > 1024$ in the case of our actual codes. An additional improvement might also be obtained by merging the Collision and Move steps in a single loop to minimize the number of load/store operations. However, neither would change matters appreciably, especially when the Collision step starts to dominate, as it does in cases of practical interest.

All the data presented so far refer to a single-processor version of the codes. Recently, we developed a parallel version of the FHP3 automaton [1] which has been run on up to six processors of the IBM 3090/600 VF under MVS/XA using the Multitasking Facility [9]. To date, the parallelization of the Move step has been obtained by dispatching the six routines which perform the propagation along the six directions of the hexagonal grid. The Collision step has been parallelized by a simple segmentation of the DO-loops which run over the computational domain, as mentioned in the previous section [15].

It is worth pointing out that, as in any parallel application, due to the parallelization costs (dispatching and synchronization) the required CPU time increases with the number of processors used. However, the benefits of parallelism have to be measured in terms of the elapsed time, the time one must wait for the completion of the application. A typical set of data referring to a 1024×1024 grid is reported in **Table 5**, in which P denotes the number of processors; the second and third columns indicate the elapsed time (ms/step) required by the Move and Collision steps, and the fourth column indicates the global (Move + Collision) speedup due to parallelization. From the data of Table 5, we see that the parallelization efficiency of the Move step is quite satisfactory for $P = 2$, 3, and 6. This is due to the fact that since the Move phase consists of six independent routines of comparable execution time, the computational load of different processors is well balanced only if $6/P$ is an integer. The parallelization efficiency of the Collision step ranges from 0.99 ($P = 2$) to 0.85 ($P = 6$) with a smooth behavior because the loop-segmentation technique guarantees a balanced computational workload. Finally, we remark that, since the Collision step requires about three times as much CPU time as the Move step, the overall speedup is satisfactory also in the case of four and five processors.

Computational efficiency of lattice gases for two-dimensional hydrodynamic analysis

As mentioned in the Introduction, the important aspect of the effectiveness of the CA approach is whether it really requires less computer resource to handle the same amount of useful information. Obviously, a serious investigation of this question would require a systematic and quantitative benchmark analysis, which lies beyond the scope of the present work. Moreover, since all of the programs we have examined thus far pertain only to the solution of "Navier–Stokes-like" equations, any extrapolation of the forthcoming considerations to other areas is unjustified.

The crucial hydrodynamic parameter to be considered is the "effective" Reynolds number, which is defined as

$$Re = G\frac{UD}{\nu}, \tag{23}$$

where D is a characteristic dimension of the geometry, U is the macroscopic speed of the fluid, ν represents its kinematic viscosity, and G designates the coefficient arising from the lack of Galilean invariance. As is known, the Reynolds number measures the degree of turbulence of the fluid via the advection/dissipation ratio, which in turn fixes the shortest wavelengths excited in the motion (dissipative scale).

More specifically, the dissipative scale d is related to the macroscopic scale by a power-law relation $d/D \sim Re^{-m}$, with $m = -0.5$ in two dimensions (Batchelor–Kraichnan theory [16], well supported by numerical simulations [17]) and $m = -0.75$ in three dimensions (Kolmogorov theory of energy cascade [18], well supported by experimental data [19]). Thus, the simplest criterion for judging the effectiveness of a lattice-gas simulation is that its mesh spacing be considerably less than the dissipative length.

In a lattice containing n sites over a length l, the maximum achievable Reynolds number is given by

$$Re_{max} = nCMD/l \equiv \alpha n, \quad (24)$$

where M is the Mach number of the flow, which must be kept smaller than one to ensure incompressibility;* C is a dimensionless coefficient that depends on the specific collision rule implemented and is typically 1–10. On the other hand, the storage requirements scale as Sn^2 and the CPU time as Sn^3, because reducing the lattice spacing causes the particles to require a correspondingly longer time to cover a given macroscopic length. If we are willing to accept this scaling, it can be inferred (more sophisticated arguments can be found in [20]) that $STORAGE = S(Re_{max}/\alpha)^2$ and $CPU = K(Re_{max}/\alpha)^3$, where K is a factor which depends on the computer performance, which, for the 3090 Vector Facility, can be assumed to be about 0.1 second per step per megasite.

These two relationships are plotted in **Figure 7**. It was assumed that $K = 0.1$ s/step/megasite, $MC = 1$, and $D/l = 0.1$. The CPU time plotted corresponds to the evolution over n time steps.

From this, it follows that in an installation which offers 256 megabytes of central memory, a Reynolds number in excess of 1000 can be reached. This is high enough for the investigation of the statistical properties of two-dimensional turbulence. However, running a lattice gas in this regime to solve the Navier–Stokes equations would require more than 100 hours of CPU time, which would be much more expensive than making use of a spectral code [21]. Runs carried out at the IBM European Center for Scientific and Engineering Computing for the purpose of studying the statistical properties of free-decaying two-dimensional turbulence [22] have indicated that a resolution of 8192^2 gridpoints is only marginally sufficient to reveal the physical behavior emerging from the spectral simulation. More precisely, this resolution has been found to yield approximately the same amount of hydrodynamic information as a 64^2 spectral run, at a computational cost which is greater by about three orders of magnitude in CPU time and two orders of magnitude in storage requirements. These figures reflect unfavorably on the lattice-gas method. However, it is important to remark that, using present-day state-of-the-art means, resorting to more efficient collision rules (such as those given by the pseudo-4D scheme described in the next section) and special-purpose hardware can reduce the computational costs of the lattice-gas simulation by a factor of about a hundred in CPU time and ten in memory requirements. These are significant improvements, especially if one considers that, since the subject is still in its scientific "infancy," major progress can be expected in both its theoretical and technological aspects.

Generally speaking, a basic difficulty is preventing the dissipation from overwhelming the convective effects which characterize the physics of highly turbulent fluids. This can be seen from **Figures 8** and **9**, which respectively depict a turbulent fluid configuration at time zero and after 16384 steps of the FHP2 automaton. The dissipation is much more visible than the advection. A similar difficulty is encountered in applications of lattice gases to problems of wave propagation [23], and it is conjectured that a breakthrough in the applicability of lattice gases will require finding some way to reduce their diffusivity, or, equivalently, to reduce the amount of microscopic noise produced by the automaton. Work in this direction, focused on the application of lattice-gas techniques to the study of flows past a cylinder, is currently in progress [24]. Even when we restrict outselves to considerations of efficiency in a general-purpose environment, the competitiveness of cellular automata modeling improves in situations characterized by irregular geometries and small values of the Reynolds number, such as those encountered, for example, in the study of flows through porous media [25]. With one-bit processor special hardware, the enhanced role played by irregular geometries favors the use of lattice gases because complex and irregular geometries can be handled easily by cellular automata rules. (It is sufficient to define different Boolean rules in the region assigned to the obstacle.)

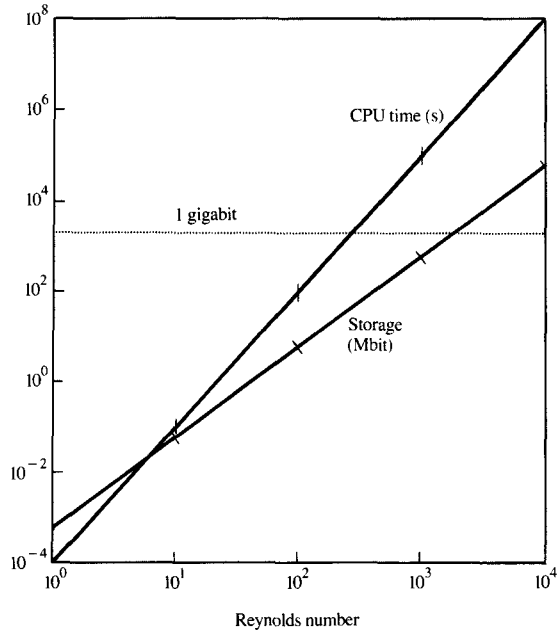

Figure 7

CPU time and storage requirements for the two-dimensional lattice-gas simulation as a function of Reynolds number.

* It follows from Equations (10), (12), and (14) that the pressure gradient driving linear-mode propagation acquires a spurious multiplicative factor $(1 - M^2)$ whose effects disappear only in the limit of incompressible flows ($M \to 0$).

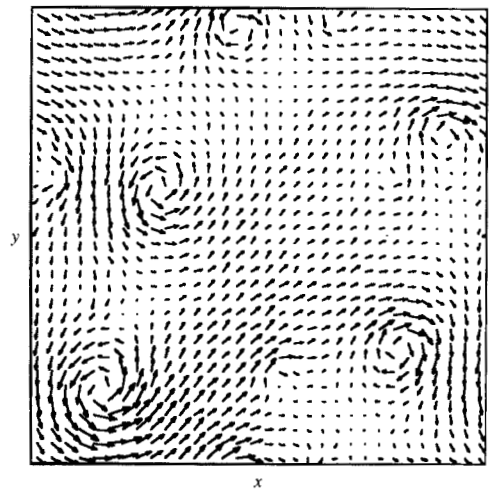

Figure 8

Initial turbulent configuration of a cellular automaton fluid. The arrows represent the macroscopic velocity along the x and y directions.

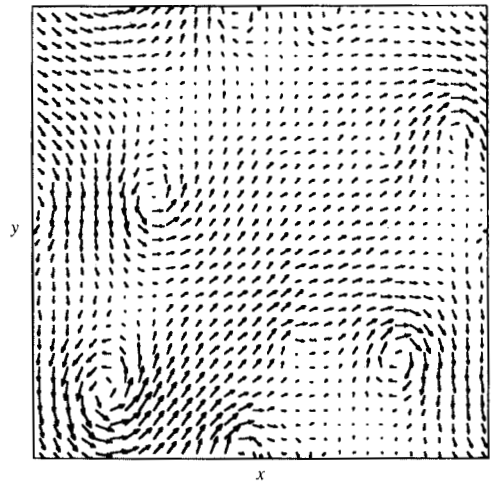

Figure 9

The turbulent configuration of Figure 8 after 16 384 automata steps.

Lattice-gas automata in three dimensions

Although we have thus far considered only two-dimensional flow, it is clearly of importance to examine whether the lattice-gas approach can be extended to three dimensions. To do so, it should just be noted that the triangular lattice was chosen for a two-dimensional gas in order to achieve isotropy of the pressure and viscous stress tensors. Unfortunately, it is known from elasticity theory that there exists no lattice in three dimensions for which general symmetrical fourth-order tensors are isotropic. From this, one might conclude that it is impossible to simulate realistic fluid mechanics in three dimensions using lattice-gas automata.

However, d'Humières, Lallemand, and Frisch [26] have noticed that a suitable lattice exists in four dimensions: a face-centered-hypercubic (FCHC) lattice. Its generating vectors connect the center of a hypercube to each of its 24 two-dimensional faces, thus defining a 24-velocity lattice-gas model. The components of the velocity vectors are deduced from one of the faces by application of permutations and sign changes of the coordinates. For example, starting from $(I, I, 0, 0)$ one generates $(I, 0, I, 0)$, $(I, 0, 0, I)$, $(0, I, I, 0)$, $(0, I, 0, I)$, $(0, 0, I, I)$, where $I = \pm 1$.

The symmetry group of the lattice has 1152 elements and is large enough to ensure the isotropy of any symmetrical fourth-order tensor. Clearly, because four-dimensional models are more demanding than three-dimensional ones for practical applications, the lattice width along the fourth dimension should be kept as small as possible. In practice, this width can contain only one node, with periodic boundary conditions along the fourth dimension. This leads to a three-dimensional model for which the usual Navier–Stokes equations are recovered in the incompressible limit. In addition, this model also leads to a fourth transport equation (for a passive scalar) which arises from momentum considerations. The FCHC lattice can be projected back into a three-dimensional cubic lattice in which the existence of the underlying fourth dimension is retained through the inclusion of the following particles:

- Twelve particles moving on the six links connecting the nodes to their nearest neighbors (bold arrows in **Figure 10**), six having a "spin" of 1 and six having a "spin" of -1.
- Twelve particles moving on the twelve links connecting the nodes to their next-nearest neighbors (light arrows in Figure 10), each having a "spin" of 0.

As stated in the previous section, the efficiency of lattice-gas automata as a numerical scheme is tightly coupled to the value of the viscosity, which should be kept as small as possible. Since each node is described by a 24-bit word, its state can assume $2^{24} = 16\,777\,216$ different configurations. Obviously, such a huge number of states precludes the possibility of the manual design of associated collision rules, as for the two-dimensional models. In this context, Henon

[27] has presented a general strategy for designing efficient collision rules which maximize the available Reynolds number. The strategy is as follows: The number of relevant states is reduced using the symmetry group so that of the 7009 possible momentum values, only 37 are returned. The states are then sorted by equivalence classes of states with the same number of particles and the same momentum (which can be exchanged during the collision step); this is accomplished by using an optimization criterion which can be applied independently to each equivalence class and leads to the least viscous model. Then the symmetry group is again applied to derive the collision rules for all of the states which are recorded in a huge lookup table of 2^{24} 24-bit words (or 48 megabytes of memory).

As for the two-dimensional case, the algorithm is based on two steps: Collision and Propagation. In the Collision step the new state of the automaton is obtained node by node by (hard-wired) indirect addressing of the huge lookup table mentioned above. This table need only be generated once; it is subsequently retrieved from mass storage before each run. The Propagation step is split into three stages corresponding to propagation along the directions x, y, and z respectively. During each of these stages, bits associated with a given propagation direction are first extracted from the computer word by masking operations and subsequently moved by address shiftings. Both the Collision and Propagation steps have been multitasked on four processors (for full details see [28]).

Using this basic algorithm, three-dimensional flows around a circular plate were simulated in a $128 \times 128 \times 256$ space [29]. Nonstationary flows were obtained at a Reynolds number of about 150 with a processing speed of about 30 million updates per second on the four processors of a CRAY2 supercomputer.

However, use of the algorithm has several drawbacks:

- A large amount of memory is required (e.g., a 256^3 lattice requires 48 + 48 megabytes, available only on very large mainframes).
- Its speed of execution is directly related to the efficiency of indirect and random addressing in a very large lookup table.
- Its lookup table causes parallel algorithms to depend on the efficiency of the computer in managing access conflicts to a common memory.

The second point has negative effects in any computing environment, and is particularly severe if use is made of a computer which relies on data in a fast memory of moderate size to achieve efficient memory access. For this reason and because of the desirability of decreasing the demands of the collision step on memory, the usefulness of this approach may be crucially dependent on further efforts to devise new algorithms based either on Boolean logic or on repeated applications of smaller lookup tables.

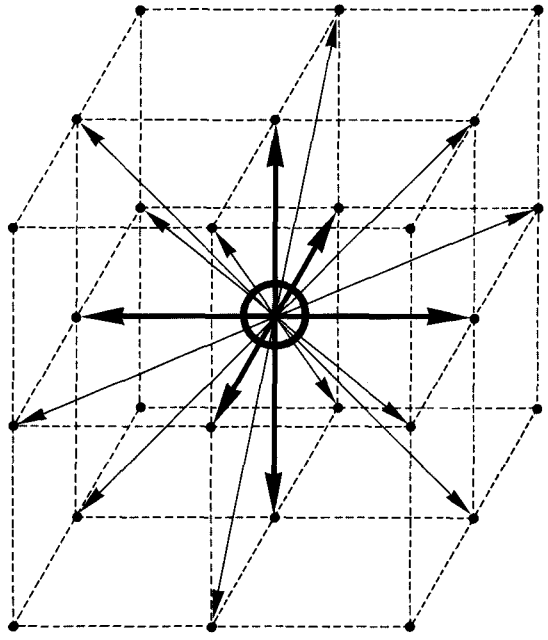

Figure 10

Three-dimensional projection of the face-centered hypercube (FCHC) used for the pseudo-4D algorithm. The bold arrows represent four-dimensional links (fourth component = ±1); the light arrows represent three-dimensional links (fourth component = 0).

Concluding remarks

The lattice-gas method provides a means for constructing a family of "synthetic" fluids on a computer and studying the passage from the microscopic to the hydrodynamic domain [30].

From an application viewpoint, it is too early to identify areas to which it will most effectively apply. Its hydrodynamic applications are limited to those associated with moderate Reynolds numbers, both in two and three dimensions. Two applications are promising: application to the study of moving boundaries between different media, such as those occurring in combustion or chemical reactions [31]; and application to the study of Brownian motion in suspensions [32] having flows with moderate Reynolds numbers and for which the simulations can take full advantage of the intrinsic noise of the lattice gas. However, study of the latter will require an improved understanding of associated solid-boundary phenomena.

Acknowledgments

We are indebted to P. Sguazzero and M. Johnson for many helpful suggestions.

References

1. U. Frisch, D. d'Humières, B. Hasslacher, P. Lallemand, Y. Pomeau, and J. P. Rivet, *Complex Syst.* **1,** 649 (1987).
2. *Complex Systems* **1** (1987), Proceedings of the workshop "Modern Approaches to Large Non-Linear Systems," Santa Fe, NM, October 1986.
3. J. Hardy, Y. Pomeau, and O. De Pazzis, *Phys. Rev. A* **13,** 1949 (1976).
4. U. Frisch, K. Hasslacher, and Y. Pomeau, *Phys. Rev. Lett.* **56,** 1065 (1986).
5. R. Gatignol, "Théorie Cynétique des Gas à Répartition Discrète de Vitesses," *Lecture Notes in Physics*, Vol. 36, Springer-Verlag, Berlin, 1975.
6. S. Chapman and T. G. Cowling, *The Mathematical Theory of Non Uniform Gases*, Cambridge University Press, Cambridge, England, 1939.
7. C. Burges and S. Zalesky, *Complex Syst.* **1,** 31 (1987).
8. M. Creutz, L. Jacobs, and C. Rebbi, *Phys. Rev. Lett.* **42,** 1390 (1979).
9. *VS Fortran Version 2: Language and Library Reference*, Order No. SC26-4221-1 (1986); available through IBM branch offices.
10. N. Margolus, T. Toffoli, and G. Vichniac, *Phys. Rev. Lett.* **56,** 1694 (1986).
11. W. Buchholz, *IBM Syst. J.* **25,** 51 (1986).
12. P. Clavin, D. d'Humières, P. Lallemand, and Y. Pomeau, *C. R. Acad. Sci. Paris* **13,** 1169 (1986).
13. F. Szelényi, Diploma Thesis, University of Linz, Austria, 1987.
14. S. Succi, *Comp. Phys. Commun.* **47,** 173 (1987).
15. S. Succi, F. Szelényi, and P. Santangelo, *Proceedings of the 2nd Conference on Vector and Parallel Computing*, Tromsö, Norway, 1988.
16. G. Batchelor, *Phys. Fluids* **56,** 1691 (1941).
17. M. E. Brachet, M. Meneguzzi, and P. L. Sulem, *Phys. Rev. Lett.* **56,** 353 (1986).
18. A. W. Kolmogorov, *C. R. Acad. Sci. USSR* **30:301,** 538 (1969).
19. A. S. Monin and A. M. Yaglom, *Statistical Fluid Mechanics*, J. L. Lumley, Ed., MIT Press, Cambridge, MA, 1975, p. 2.
20. S. Orszag and V. Yakhot, *Phys. Rev. Lett.* **56,** 1691 (1986).
21. R. Benzi, P. Santangelo, and S. Patarnello, *Europhys. Lett.* **7,** 811 (1987).
22. S. Succi, P. Santangelo, and R. Benzi, *Phys. Rev. Lett.* **60,** 2738 (1988).
23. C. Sword, *Stanford Exploration Project Technical Report No. 56*, pp. 1–18, 1987.
24. S. Succi, R. Benzi, and F. Higuera, ECSEC preprint, to appear in the proceedings of the workshop on "Discrete Kinetic Theory, Lattice Gas Dynamics and Foundations of Hydrodynamics," Turin, Italy, September 1988.
25. D. H. Rothmann, *Geophys.* **53,** 509 (1988).
26. D. d'Humières, P. Lallemand, and U. Frisch, *Europhys. Lett.* **2,** 291 (1987).
27. M. Henon, *Complex Syst.* **1,** 475, 763 (1987).
28. J. P. Rivet, Ph.D. Thesis, University of Nice, France, 1988.
29. J. P. Rivet, M. Henon, U. Frisch, and D. d'Humières, *C. R. Acad. Sci. Paris* **2,** 751 (1987); *Europhys. Lett.* **7,** No. 3, 231–237 (1988).
30. S. Succi, R. Benzi, and P. Santangelo, *J. Phys. A: Math. Gen.* **21,** L771–L776 (1988).
31. P. Clavin, P. Lallemand, Y. Pomeau, and G. Searby, *J. Fluid Mech.* **188,** 437 (1988).
32. A. Ladd, M. Colvin, and D. Frenkel, *Phys. Rev. Lett.* **60,** 975 (1988).

Received April 6, 1988; accepted for publication October 12, 1988

Sauro Succi *IBM Italy, European Center for Scientific and Engineering Computing, Via Giorgione 159, 00147 Rome, Italy.* Dr. Succi received his doctorate in nuclear engineering from the University of Bologna in 1979 and a Ph.D. in plasma physics from the Swiss Federal Institute of Technology in 1987. In 1980, as a member of the Department of Fast Breeder Reactors of the National Committee of Nuclear Energy (CNEN) in Bologna, he developed algorithms for the optimal control of nuclear accidents. Dr. Succi subsequently joined the Neutral Injection Team of the Max-Planck Institüt für Plasmaphysik in Garching (FRG), working on Monte Carlo simulation codes for the heating of thermonuclear plasmas. In 1982 he joined the Plasma Physics Center of the Swiss Federal Institute of Technology in Lausanne, where he was involved in the utilization and development of finite-element codes for plasma magnetohydrodynamics and plasma kinetics. In 1986 Dr. Succi joined the IBM European Center for Scientific and Engineering Computing. His current research interests include the lattice-gas modeling of fluid turbulence and the modeling of plasma turbulence, each on the IBM 3090 Vector Facility.

Dominique d'Humières *Ecole Normale Supérieure de Paris, Rue de Lhomond 24, 75231, Paris Cedex 5, France.* Dr. d'Humières graduated from the Ecole Normale Superiéure, and in 1980 received his Doctorat ès Sciences in physics from the University of Paris. Since 1974 he has been a staff member of the French Research Council (CNRS), working on problems in fluid mechanics and related areas. In 1982 and 1983 he was a visiting scholar at Stanford University and a consultant at the Xerox Palo Alto Research Center. Dr. d'Humières has made extensive contributions to the field of lattice-gas hydrodynamics. He is one of the originators of the lattice-gas "pseudo-4D" model for use in solving three-dimensional hydrodynamic problems.

Ferenc Szelényi *IBM Italy, European Center for Scientific and Engineering Computing, Via Giorgione 159, 00147 Rome, Italy; and University of Innsbruck, Innrain 52, 6020 Innsbruck, Austria.* Mr. Szelényi is currently working on tools for parallel programming at the IBM European Center for Scientific and Engineering Computing. Concurrently, he is a Ph.D. candidate at the University of Innsbruck, Austria. He previously obtained a Master's degree in computer science from the University of Linz, Austria, focusing on parallelism in FORTRAN programs.